Modelica extension for PDE

June 20, 2013

Space & coordinates

What should be specified

- Dimension of the problem (1,2 or 3D)
- ?? Coordinates (cartesian, cylindrical, spherical ...) – where this information will be used (if at all):
  - in differential operators as grad, div, rot etc.
  - in visualization of results
  - ?? in computation – perhaps equations should be transformed and the calculation would be performed in cartesian coordinates
- Names of independent (coordinate) variables (x, y, z, r, φ, θ...)

Domain & boundary

What should be specified

- the domain where we perform the computation and where equations hold
- boundary and its subsets where particular boundary conditions hold
- normal vector of the boundary

Possible approaches

Parametrization of the domain with shape function and ranges – from The Book (Principles of ...), section 8.5.2

Example from the book:

```
model HeatCircular2D
        import DifferentialOperators2D.*;
        parameter DomainCircular2DGrid omega;
        FieldCircular2DGrid u(domain=omega, FieldValueType=SI.Temperature);
equation
        der(u) = pder(u,D.x2)+ pder(u,D.y2)      in omega.interior;
        nder(u) = 0                 in omega.boundary;
end HeatCircular2D;

record DomainCircular2DGrid "Domain being a circular region"
        parameter Real radius = 1;
        parameter Integer nx = 100;
        parameter Integer ny = 100;
```

```
        replaceable function shapeFunc = circular2Dfunc "2D circular region";
        DomainGe2D interior(shape=shapeFunc,range={{0,radius},{0,1}},geom= ... )
        DomainGe2D boundary (shape=shapeFunc, range={{radius, radius), { 0,1}} ,
        function shapeFunc = circular2Dfunc "Function spanning circular region";
end DomainCircular2DGrid;

function circular2Dfunc "Spanned circular region for v in range 0..1"
        input Real r,v;
        output Real x,y;
algorithm
        x := r*cos (2*PI*v);
        y := r*sin(2*PI*v);
end circular2Dfunc;

record FieldCircular2DGrid
        parameter DomainCircular2DGrid domain;
        replaceable type FieldValueType = Real;
        replaceable type FieldType = Real[domain.nx,domain.ny,domain.nz];
        parameter FieldType start = zeros(domain.nx,domain.ny,domain.nz.);
        FieldType Val;
end FieldCircularZDGrid;
```

And modified version, where all numerical stuff (grid, number of points – this should be configured using simulation setup or annotations) omitted, modified `pder` operator, `Field` as Modelica build-in type:

```
model HeatCircular2D
        parameter DomainCircular2D omega;
        Field Real u(domain=omega, start = 0, FieldValueType=SI.Temperature);
equation
        pder(u,time) = pder(u,x)+ pder(u,y) in omega.interior;
        pder(u,omega.boundary.n) = 0    in omega.boundary;
end HeatCircular2D;

record DomainCircular2D "Domain being a circular region"
        parameter Real radius = 1;
        function shapeFunc = circular2Dfunc "Function spanning circular region";
        DomainGe2D interior(shape=shapeFunc, range={{0,radius},{0,1}});
        DomainGe2D boundary (shape=shapeFunc, range={{radius, radius), {0,1}});
end DomainCircular2DGrid;

function circular2Dfunc "Spanned circular region for v in range 0..1"
        input Real r,v;
        output Real x,y;
algorithm
```

```
        x := r*cos(2*PI*v);
        y := r*sin(2*PI*v);
end circular2Dfunc;
```

Description by the boundary Domain is defined by closed boundary curve, which may by composed of several connected curves. Needs new operator *interior* and type *Domain2d* (and *Domain1D* and *Domain3d*). (similarly used in FlexPDE – http://www.pdesolutions.com/.)

```
package BoundaryRepresentation
  partial function cur
    input Real u;
    output Real x;
    output Real y;
  end cur;
  function arc
    extends cur;
    parameter Real r;
    parameter Real cx;
    parameter Real cy;
  algorithm
    x:=cx + r * cos(u);
    y:=cy + r * sin(u);
  end arc;
  function line
    extends cur;
    parameter Real x1;
    parameter Real y1;
    parameter Real x2;
    parameter Real y2;
  algorithm
    x:=x1 + (x2 - x1) * u;
    y:=y1 + (y2 - y1) * u;
  end line;
  function bezier3
    extends cur;
    //start-point
    parameter Real x1;
    parameter Real y1;
    //end-point
    parameter Real x2;
    parameter Real y2;
    //start-control-point
    parameter Real cx1;
    parameter Real cy1;
```

```
  //end-control-point
  parameter Real cx2;
  parameter Real cy2;
algorithm
  x:=(1 - u) ^ 3 * x1 + 3 * (1 - u) ^ 2 * u * cx1 + 3 *
      (1 - u) * u ^ 2 * cx2 + u ^ 3 * x2;
  y:=(1 - u) ^ 3 * y1 + 3 * (1 - u) ^ 2 * u * cy1 + 3 *
      (1 - u) * u ^ 2 * cy2 + u ^ 3 * y2;
end bezier3;
record Curve
  function curveFun = line;
  // to be replaced with another fun
  parameter Real uStart;
  parameter Real uEnd;
end Curve;
record Boundary
  constant Integer NCurves;
  Curve curves[NCurves];
  //     for i in 1:(NCurves-1) loop
  //assert(Curve[i].curveFun(Curve[i].uEnd) = Curve[i
      +1].curveFun(Curve[i+1].uStart), String(i)+"th
      curve and "+String(i+1)+"th curve are not
      connected.",level = AssertionLevel.error);
  //     end for;
  //     assert(curves[NCurves].curveFun(curves[NCurves
      ].uEnd) =
  //
      curves[1].curveFun(curves[1].uStart),
  //
      String(NCurves)+"th curve and first curve are not
      connected.",
  //
      level = AssertionLevel.error);
end Boundary;
record DomainHalfCircle
  constant Real pi = Modelica.Constants.pi;
  arc myArcFun(cx = 0, cy = 0, r = 1);
  Curve myArc(curveFun = myArcFun, uStart = pi / 2,
      uEnd = (pi * 3) / 2);
  line myLineFun(x1 = 0, y1 = -1, x2 = 0, y2 = 1);
  Curve myLine(curveFun = myLineFun, uStart = 0, uEnd =
      1);
  line myLine2(curveFun = line(x1 = 0, y1 = -1, x2 = 0,
      y2 = 1), uStart = pi / 2, uEnd = (pi * 3) / 2);
  Boundary b(NCurves = 2, curves = {myArc,myLine});
  //new externaly defined type Domain2D and operator
```

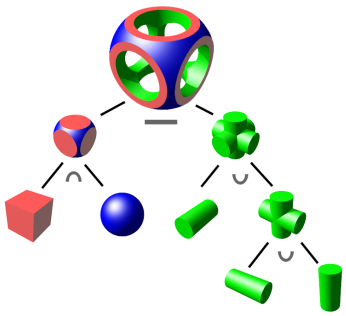

Figure 1: constructive solid geometry

```
        interior:
    Domain2D d = interior Boundary;
  end DomainHalfCircle;
end BoundaryRepresentation;
```

Listing 1: Description by boundary – example

Constructive solid geometry used in Matlab PDE toolbox, http://en.wikipedia.org/wiki/Constructive_soli

Domain is build from primitives (cuboids, cylinders, spheres, cones, user defined shapes ...) applying boolean operations *union*, *intersection* and *difference*.

Listing of points – export from CAD

Inequalities

Boundary representation (BRep) (NETGEN, STEP)

Fields

Partial derivative

$\frac{\partial^2 u}{\partial x \partial y}$ ... `pder(u,x,y)`
directional derivative ... `pder(u,n)`

Equations & boundary conditions